**Assignment II: CUDA Basics**

Franz Kaschner

28.11.2022

Link to Github repository: <https://github.com/TheKastenkarl/DD2360HT22>

**Exercise 1 - Your first CUDA program and GPU performance metrics**

1. **Explain how the program is compiled and run.**

Compiling:

!nvcc -arch=sm\_75 ./lab3\_ex1.cu -o lab3\_ex1

Running:

!./lab3\_ex1 <vector-length>

Profiling:

!/usr/local/cuda-11/bin/nv-nsight-cu-cli ./lab3\_ex1 <vector-length>

1. **For a vector length of N:**
   1. **How many floating operations are being performed in your vector add kernel?**

N floating point operations (additions).

* 1. **How many global memory reads are being performed by your kernel?**

2N global memory reads for the summands.

1. **For a vector length of 1024:**
   1. **Explain how many CUDA threads and thread blocks you used.**

In total, 1024 threads are required as one thread for every addition is created. I use 64 threads per block and ⌈(1024+64-1)/64⌉ = 16 blocks.

* 1. **Profile your program with Nvidia Nsight. What Achieved Occupancy did you get? You might find** [**https://docs.nvidia.com/nsight-compute/NsightComputeCli/index.html#nvprof-metric-comparison**](https://docs.nvidia.com/nsight-compute/NsightComputeCli/index.html#nvprof-metric-comparison) **useful.**

The measured Achieved Occupancy is 6.19%.

1. **Now increase the vector length to 131070:**
   1. **Did your program still work? If not, what changes did you make?**

Yes, it still worked.

* 1. **Explain how many CUDA threads and thread blocks you used.**

In total, 131070 threads are required as one thread for every addition is created. I use 128 threads per block and ⌈(131070+128-1)/128⌉ = 1024 blocks. Hence, not all threads are really required because we get 128 \* 1024 = 131072 threads in total.

* 1. **Profile your program with Nvidia Nsight. What Achieved Occupancy do you get now?**

The measured Achieved Occupancy is 78.04%.

1. **Further increase the vector length (try 6-10 different vector length), plot a stacked bar chart showing the breakdown of time including (1) data copy from host to device (2) the CUDA kernel (3) data copy from device to host. For this, you will need to add simple CPU timers to your code regions.**

I use nvprof for measuring the times as it is simpler and more accurate (<https://stackoverflow.com/questions/30371030/understanding-cuda-profiler-output-nvprof>).

Command: nvprof ./lab3\_ex1 <vector-length>

Chart, bar chart

Description automatically generated

**Exercise 2 - 2D Dense Matrix Multiplication**

1. **Name three applications domains of matrix multiplication.**

Robotics, Computer Vision, Neural Networks

1. **How many floating operations are being performed in your matrix multiply kernel?**

2 \* numARows \* numBColumns \* numAColumns

Times 2 because for every element you have to do an addition and a multiplication (<https://math.stackexchange.com/questions/3512976/proof-of-of-flops-in-matrix-multiplication>).

1. **How many global memory reads are being performed by your kernel?**

2 \* numARows \* numBColumns \* numAColumns + numARows \* numBColumns (if a temporal variable is used)

3 \* numARows \* numBColumns \* numAColumns (if no temporal variable is used)

1. **For a matrix A of (128x128) and B of (128x128):**
   1. **Explain how many CUDA threads and thread blocks you used.**

In total, 128 \* 128 = 16384 threads are required as one thread for every element of the output matrix C is created. I use 8 \* 8 = 64 threads per block and ⌈(128+8-1)/8⌉ \* ⌈(128+8-1)/8⌉ = 16\*16 = 256 blocks.

* 1. **Profile your program with Nvidia Nsight. What Achieved Occupancy did you get?**

The measured Achieved Occupancy is 24.21%.

1. **For a matrix A of (511x1023) and B of (1023x4094):**
   1. **Did your program still work? If not, what changes did you make?**

Yes, it still worked.

* 1. **Explain how many CUDA threads and thread blocks you used.**

In total, 511 \* 4094 = 2092034 threads are required as one thread for every element of the output matrix C is created. I use 16 \* 16 = 256 threads per block and ⌈(511+16-1)/16⌉ \* ⌈(4094+16-1)/16⌉ = 32\*256 = 8192 blocks. Hence, not all threads are really required because we get 256 \* 8192 = 2097152 threads in total.

* 1. **Profile your program with Nvidia Nsight. What Achieved Occupancy do you get now?**

The measured Achieved Occupancy is 95.86%.

1. **Further increase the size of matrix A and B, plot a stacked bar chart showing the breakdown of time including (1) data copy from host to device (2) the CUDA kernel (3) data copy from device to host. For this, you will need to add simple CPU timers to your code regions. Explain what you observe.**

I use nvprof for measuring the times as it is simpler and more accurate (<https://stackoverflow.com/questions/30371030/understanding-cuda-profiler-output-nvprof>).

Command: nvprof ./lab3\_ex2 <vector-length>

Chart, bar chart

Description automatically generated

It can be observed that most of the time Is required for the CUDA kernel. Furthermore, you can see that the fraction of execution time of the CUDA kernel increases with increasing matrix sizes. This is because the number of FLOP increase with O(numARows \* numBColumns \* numAColumns) whereas the number of elements to copy between host H and device D only increase with O(numARows \* numAColumns + numBRows \* numBColumns) for H2D and with O(numCRows \* numCColumns) for D2H.

1. **Now, change DataType from double to float, re-plot the a stacked bar chart showing the time breakdown. Explain what you observe.**

Bar chart using floats:

Chart, bar chart

Description automatically generated

The major difference is that the total time is much lower, approximately by factor 2. This is because a float only has half the size of a double.

**Exercise 3 - Histogram and Atomics**

1. **Describe all optimizations you tried regardless of whether you committed to them or abandoned them and whether they improved or hurt performance.**
2. **Which optimizations you chose in the end and why?**
3. **How many global memory reads are being performed by your kernel? Explain**
4. **How many atomic operations are being performed by your kernel? Explain**
5. **How much shared memory is used in your code? Explain**
6. **How would the value distribution of the input array affect the contention among threads? For instance, what contentions would you expect if every element in the array has the same value?**
7. **Plot a histogram generated by your code and specify your input length, thread block and grid.**
8. **For a input array of 1024 elements, profile with Nvidia Nsight and report Shared Memory Configuration Size and Achieved Occupancy. Did Nvsight report any potential performance issues?**

**Exercise 4 - A Particle Simulation Application**

1. **Describe the environment you used, what changes you made to the Makefile, and how you ran the simulation.**
2. **Describe your design of the GPU implementation of mover\_PC() briefly.**
3. **Compare the output of both CPU and GPU implementation to guarantee that your GPU implementations produce correct answers.**
4. **Compare the execution time of your GPU implementation with its CPU version.**